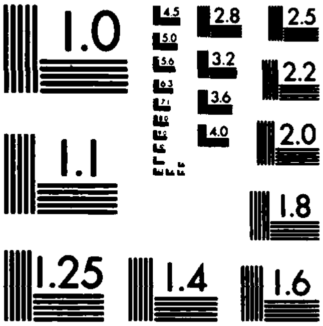
AD-A132 482 PROOF CHECKING THE RSA (RIVEST SHAMIR AND ADLEMAN) 1/1
PUBLIC KEY ENCRYPTION..(U) TEXAS UNIV AT AUSTIN INST
FOR COMPUTING SCIENCE AND COMPUTER A..
UNCLASSIFIED R S BOYER ET AL. SEP 82 ICSCA-CMP-33 F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ADA132482

12

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
ICSCA-CMP-33

2. GOVT ACCESSION NO.
AD-A132482

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Proof Checking the RSA Public Key Encryption Algorithm

5. TYPE OF REPORT & PERIOD COVERED
Technical

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Robert S. Boyer and J Strother Moore

8. CONTRACT OR GRANT NUMBER(s)
MCS-8202943
N00014-81-K-0634

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Institute for Computing Science and Computer Applications/ University of Texas at Austin
Main Building 2100
Austin, Texas 78712

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
NR 049-500

11. CONTROLLING OFFICE NAME AND ADDRESS
Software Systems Science
National Science Found.
Washington, D.C. 20550

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

12. REPORT DATE
September 1982

13. NUMBER OF PAGES
24

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Reproduction in whole or in part is permitted for any purposes of the United States government.

...roved
for p... ...; its
distrib... ...

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

18. SUPPLEMENTARY NOTES

SEP 1 1983
E

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

automatic theorem-proving, Fermat's theorem, number theory, pigeon hole principle

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We describe the use of a mechanical theorem-prover to check the published proof of the invertibility of the public key encryption algorithm of Rivest, Shamir and Adleman: $(M^e \bmod n)^d \bmod n = M$, provided n is the product of two distinct primes p and q, $M<n$, and e and d are multiplicative inverses in the ring of integers modulo $(p-1)*(q-1)$. Among the lemmas proved mechanically and used in the main proof are many familiar theorems of number theory, including Fermat's theorem: $M^{p-1} \bmod p = 1$, when $p \nmid M$. The axioms underlying the proofs are those of Peano arithmetic and ordered pairs.

83 09 15 009 ~~82~~

DTIC FILE COPY

Technical Report No. ICSCA-CMP-33
Grant No. MCS-8202943
Contract No. N00014-81-K-0634; NR 049-500

PROOF CHECKING THE RSA PUBLIC KEY ENCRYPTION ALGORITHM

Robert S. Boyer and J Strother Moore
Institute for Computing Science and Computer Applications
The University of Texas at Austin
Austin, TX 78712

September, 1982

Technical Report

Reproduction in whole or in part is permitted for any purpose of
the United States government.

Prepared for:

National Science Foundation
Software Systems Science Program
Washington, D.C. 20550

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

DTIC
COPY
INSPECTED
2

Accession For
X
A

Abstract

We describe the use of a mechanical theorem-prover to check the published proof of the invertibility of the public key encryption algorithm of Rivest, Shamir and Adleman: $(M^{e} \text{ mod } n)^{d} \text{ mod } n = M$, provided n is the product of two distinct primes p and q, $M<n$, and e and d are multiplicative inverses in the ring of integers modulo $(p-1)*(q-1)$. Among the lemmas proved mechanically and used in the main proof are many familiar theorems of number theory, including Fermat's theorem: $M^{p-1} \text{ mod } p = 1$, when $p \nmid M$. The axioms underlying the proofs are those of Peano arithmetic and ordered pairs.

Key Words: automatic theorem-proving, Fermat's theorem, number theory, pigeon hole principle.

PROOF CHECKING THE RSA PUBLIC KEY ENCRYPTION ALGORITHM

> The development of mathematics toward greater precision has led, as is well known, to the formalization of large tracts of it, so that one can prove any theorem using nothing but a few mechanical rules.
> -- Godel [4]

> But formalized mathematics cannot in practice be written down in full, and therefore we must have confidence in what might be called the common sense of the mathematician ... We shall therefore very quickly abandon formalized mathematics ... -- Bourbaki [1]

1. Introduction

A formal mathematical proof is a finite sequence of formulas, each element of which is either an axiom or the result of applying one of a fixed set of mechanical rules to previous formulas in the sequence. It is thus possible to check mechanically whether a given sequence is a formal proof. However, formal proofs are rarely used. Instead, typical proofs in journal articles, textbooks, and day-to-day mathematical communication use informal notation and leave many of the steps to the reader's imagination. Nevertheless, by transcribing the sentences of the proof into a formal notation, it is sometimes possible to use today's automatic theorem-provers to fill in the gaps between published steps and thus mechanically check some published, informal proofs.

In this paper we illustrate this idea by mechanically checking the recently published proof of the invertibility of the public key encryption algorithm, described by Rivest, Shamir, and Adleman [10]. We will briefly explain the idea of public key encryption to motivate the theorem proved.

In [10] a mathematical function, here called CRYPT, is defined. CRYPT(M,e,n) is the encryption of message M with key (e,n). The function has the following important properties:

1. It is easy to compute CRYPT(M,e,n).

2. CRYPT is "invertible", i.e., if M is encrypted with key (e,n) and then decrypted with key (d,n) the result is M. That is, CRYPT(CRYPT(M,e,n),d,n) = M, under suitable conditions on M, n, e and d.

3. Publicly revealing CRYPT and (e,n) does not reveal an easy way to compute (d,n). Public key encryption thus avoids the problem of distributing keys via secure means. Each user (e.g., a computer on a network) generates an encryption key and a corresponding decryption key, publicizes the encryption key to enable others to send private messages, and never distributes the decryption key.

The function defined in [10] is CRYPT(M,e,n) = M^e mod n; in addition, algorithms are given for constructing e, d, and n so that CRYPT has the three properties above. The first two properties are proved in [10]. The third property is not proved; instead the authors of [10] argue that "all the obvious approaches to breaking our system are at least as difficult as factoring n." Since there is no known algorithm for efficiently factoring large composites, the security property of CRYPT is obtained by constructing n as the product of two very large (200 digit) primes.

In this paper we focus on mechanically checking the proofs of the first two properties. A precise statement of the "invertibility" property is: CRYPT(CRYPT(M,e,n),d,n) = M, if n is the product of two distinct primes p and q, M<n, and e and d are multiplicative inverses in the ring of integers modulo (p-1)*(q-1). Our mechanical proof of this theorem requires that we first prove many familiar theorems of number theory, including Fermat's theorem: M^{p-1} mod p = 1, when $p \not| M$.

2. An Experiment with a Journal Proof Checker

The theorem-prover we use is the current version of the system described in [2]. The theorem-prover deals with a quantifier free first order logic providing equality, recursive definition, mathematical induction, and inductively constructed objects such as the natural numbers and binary trees (ordered pairs).

The theorem-prover is a large computer program that orchestrates the application of many proof techniques (e.g., simplification and mathematical induction) under heuristic control. The theorem-prover, as of May, 1978, is completely described in [2]. Improvements made to the system since the publication of [2] include the addition of linear arithmetic and equality decision procedures, the extension of the definitional principle to include reflexive functions as described in [9], and a metafunction facility permitting the incorporation of new simplifiers after they have been mechanically proved correct [3]. Finally, we have added a primitive "hint" facility so that the user can tell the theorem-prover to "use lemma x with instantiation y" or to "induct according to schema z".

The theorem-prover is automatic in the sense that once it begins a proof attempt, no user guidance is permitted. However, every time it accepts a definition or proves a theorem it stores the definition or theorem for future use. By presenting the theorem-prover with an appropriate sequence of lemmas to prove, the user can "lead" it to proofs it would not otherwise discover. Thus, the distinction between a proof checker and an automatic theorem-prover blurs once the system remembers and uses previously proved facts. An automatic theorem-prover merely enables the user to leave out some of the routine proof steps. A sufficiently good automatic theorem-prover might enable the user to check an "informal" proof by presenting to the machine no more material than one would present to a human colleague.

The material contained in boxes in this paper represents material typed by the user and checked by the theorem-prover. Our claim is that the boxed material very closely resembles traditional "informal" proofs. To make this more obvious to readers unfamiliar with our formal notation, we have taken the liberty of transcribing the user type-in into conventional mathematical

English. However, in section 6 we give the actual user type-in and each English sentence in boxed material indicates the corresponding numbered "event" in section 6.

When we began the encryption proofs we initialized the theorem-prover to a library containing several hundred previously proved theorems. Most of the theorems in this library were irrelevant to the encryption proofs (e.g., there are many theorems about list processing functions such as REVERSE, FLATTEN, and SORT). However, among the theorems in the library are many elementary facts about addition, multiplication, and integer division with remainder. The library concludes with the theorem that prime factorizations are unique.

3. Correctness of CRYPT

To show that M^e mod n is easy to compute -- even when the numbers involved contain hundreds of digits -- Rivest, Shamir and Adleman exhibit an algorithm for computing it in order $\log_2(e)$ steps. Below we define CRYPT as a recursive version of their algorithm and prove that it computes the desired function. The notation e/2 below denotes integer division, i.e., the floor of the rational quotient.

Box 1

We define the encryption algorithm as the recursive function CRYPT (event 1):

```
CRYPT(M,e,n)
  =
if e is not a number or is 0,
   then 1;
elseif e is even,
   then
   $(\text{CRYPT}(M,e/2,n))^2$ mod n;
   else
   $(M * (\text{CRYPT}(M,e/2,n)^2 \bmod n)) \bmod n$.
```

Observe (events 2 and 3) that:

Theorem. TIMES.MOD.1:
(x*(y mod n)) mod n = (x*y) mod n

Theorem. TIMES.MOD.2:
(a*(b*(y mod n))) mod n = (a*(b*y)) mod n.

The second follows immediately from the first by replacing x with (a*b).

Then CRYPT computes M^e mod n, provided n is not 1:

Theorem. CRYPT.CORRECT:
N≠1 -> CRYPT(M,e,n) = M^e mod n.

This completes the proof that the algorithm computes the function claimed in [10]. In order to reinforce in the reader's mind the fact that the theorem-prover assents to these claims only after proving them we offer the following comments.

Before accepting a proposed definition as a new axiom, the theorem-prover checks that it has certain properties. The properties guarantee that there exists one and only one function satisfying the defining equation. The most interesting of these properties is that the recursion in the proposed definition is well-founded. To accept CRYPT the theorem-prover guesses that e decreases in each step and then proves it by showing that when e is a non-0 number, e/2 is strictly smaller than e.

Observe that CRYPT is simply the so-called "binary method" of computing M^e (see [8]), except that all multiplications are done modulo n. Thus, TIMES.MOD.1 is obviously an important lemma in the correctness of CRYPT. We had the system prove it before even considering the mechanical proof of CRYPT.CORRECT.

The key to the theorem-prover's proof of TIMES.MOD.1 is a lemma in the standard library stating that if y is any natural number and n is positive then y can be expressed as r+qn, where r<n (see pg. 243 of [2] for the system's inductive proof). To prove TIMES.MOD.1, the system considers the cases on whether y is a natural and n is positive. If both are true, the system replaces y with r+qn and applies the lemmas that multiplication distributes over addition and that i+jn mod n is i mod n. The "pathological" cases where either y is not a natural or n is not positive yield trivially. The proof requires about 23 seconds of cpu time.

TIMES.MOD.2 is just an instance of TIMES.MOD.1 to which the associativity of multiplication has been applied to an inner term. Since TIMES.MOD.2 is uninteresting to a human proof checker, the need for it in our mechanical proof exposes a deficiency in our mechanical proof checker. Why is this line needed by the machine? In our system, TIMES.MOD.1 and (i*j)*k = i*(j*k) are used only as rewrite rules from left to right. Consider the term ((a*b)*(y mod n)) mod n. If TIMES.MOD.1 is applied first and associativity is applied to the result, the term is simplified to (a*(b*y)) mod n. But if associativity is applied first, TIMES.MOD.1 no longer applies.[1] TIMES.MOD.2 just equates the two different simplifications and makes it unimportant in this instance whether associativity is applied first.

How was the need for TIMES.MOD.2 discovered? We certainly did not anticipate it. Instead, immediately after proving TIMES.MOD.1 we thought the machine could prove CRYPT.CORRECT. We commanded it to do so and watched its proof attempt on the screen. (Imagine watching a colleague proving the theorem on the blackboard.) We saw the term (a*(b*(y mod n))) mod n arise and remain "unsimplified" even though we knew it was (a*(b*y)) mod n. At that point we interjected with TIMES.MOD.2.

We now consider the proof of CRYPT.CORRECT. The first time we submitted CRYPT.CORRECT we did not include the hypothesis that $N \neq 1$, because the hypothesis is not noted by Rivest, Shamir and Adleman, who imply that the algorithm always computes M^e mod n. However, the theorem-prover failed to prove the simpler conjecture and exhibited a formula showing that the encryption algorithm does not compute M^e mod n when e is 0 and n is 1. In practice, n is always larger than 1, so the additional hypothesis is no burden.

The theorem-prover proves CRYPT.CORRECT by inducting on e, taking for the base case that e is nonnumeric or 0, and taking for the induction step that e is positive and that the conjecture holds for e/2. The proof required about 6 minutes of cpu time. Note that no inductive invariants were supplied.

[1] This is the Knuth-Bendix problem in rewrite driven simplification. See [6] for an elegant solution to the problem in certain cases.

4. Fermat's Theorem

The proof of the invertibility of CRYPT in [10] assumes the reader is familiar with elementary number theory up through Fermat's theorem. While a production model "journal proof checker" would come factory equipped with a good number theory library, we had no such library when we began the encryption proofs. We therefore had the system prove the following theorems:

- Many elementary facts about remainder and exponentiation (events 5-17).

- Theorem 53 of Hardy and Wright's *An Introduction to the Theory of Numbers* (event 17): Suppose p and q are distinct primes, a mod p = b mod p, and a mod q = b mod q. Then a mod p*q = b mod p*q. Hence (event 18), under the additional hypothesis b<p*q, a mod p*q = b.

- Theorem 55 of Hardy and Wright (event 19): Suppose p is a prime and p does not divide M. Then M*x mod p = M*y mod p iff x mod p = y mod p. Hence (event 20), by letting y be 1, if p is a prime, M*x mod p = M mod p iff either p|M or x mod p = 1.

- The Pigeon Hole Principle (events 21 through 34): If L is a sequence of length n, every element of L is a positive number, no element occurs twice in L, and every element of L is less than or equal to n, then L is a permutation of the sequence [n n-1 ... 2 1].

- The following straightforward observations about permutations and the concept of the product over (the elements of) a sequence:

    - (Event 35) If L1 is a permutation of L2 then the product over L1 is equal to that over L2.

    - (Event 36) The product over [n n-1 ... 2 1] is n!.

    - (Event 37) Hence, if L is a permutation of [n n-1 ... 2 1] then the product over L is n!.

    - (Event 38) If p is a prime and n<p, then p does not divide n!.

We then had the theorem-prover check the proof of Fermat's theorem in [7].

Box 2

Let S(n,M,p) be the sequence [n*M mod p, (n-1)*M mod p, ..., 1*M mod p] (event 39).

In the text below we make the convention that p is a prime that does not divide M.

(Event 40) The product over S(n,M,p) mod p is equal to n!*M^n mod p.

Observe (event 41) that if i<j<p then j*M mod p is not a member of S(i,M,p) (Hint: induct on i). Hence (event 42), if n<p, then no element of S(n,M,p) occurs twice. Furthermore, each element of S(n,M,p) is positive, each is less than or equal to p-1, and there are n elements (events 43, 44, and 45).

Thus, from the Pigeon Hole Principle we have that the product over S(p-1,M,p) is (p-1)!. But we have (from event 40) that the product over S(p-1,M,p) mod p is (p-1)!*M^{p-1} mod p. Hence, Fermat's theorem (event 46).

5. Invertibility of CRYPT

We now prove that CRYPT(CRYPT(M,e,n),d,n) = M, if n is the product of two distinct primes p and q, M<n, and e and d are multiplicative inverses in the ring of integers modulo (p-1)*(q-1).

Box 3

From Fermat's theorem we get (event 47)

$$(M*M^{k*(p-1)}) \text{ mod } p = M \text{ mod } p$$

for all primes p (regardless of whether p divides M). Thus, if p and q are prime, we get (by taking two instantiations of event 47) (events 48 and 49):

$$(M*M^{k*(p-1)*(q-1)}) \text{ mod } p = M \text{ mod } p$$

and

$$(M*M^{k*(p-1)*(q-1)}) \text{ mod } q = M \text{ mod } q$$

Thus (event 50), if p and q are distinct primes, M is a number less than p*q, and x mod (p-1)*(q-1) is 1, then

$$M^{x} \text{ mod } p*q = M.$$

Hence (event 51), if p and q are distinct primes, n is p*q, M is a number less than n and e*d mod (p-1)*(q-1) is 1, CRYPT(CRYPT(M,e,n),d,n) = M. Q.E.D.

6. Input to the Theorem-Prover

We here present the sequence of definitions and theorems typed to the theorem-prover. Over 180 definitions and lemmas are involved in the dependency tree from CRYPT.INVERTS down to the axioms of Peano arithmetic and sequences. Most of the necessary facts were in the library to which we initialized the system when we began the proofs of CRYPT. Fifty-one events had to be typed to lead the system from its initial library to CRYPT.INVERTS. These events include the proofs of Theorems 53 and 55 of [5], the Pigeon Hole Principle, and Fermat's theorem. After informally describing the initial library we list the fifty-one events typed specifically for these proofs. The reader can gain an understanding of the theorem-prover's contribution to the proofs by proving each of the theorem's below.

The syntax used here is the prefix syntax used as our system's formal language.

6.1. The Initial Library

The initial library was a revised and extended version of the list of events in Appendix A of [2], containing over 400 definitions and proved theorems. The library contains the system's proof of the unique prime factorization theorem, as well as many irrelevant lemmas (e.g., the correctness of a tautology checker). The new version of that library contains several generally useful arithmetic theorems about REMAINDER and EXP, added since [2] was published. Among those "new" theorems used in the proofs reported here are:

```
Theorem.  REMAINDER.PLUS.TIMES.1 (rewrite):
(EQUAL (REMAINDER (PLUS X (TIMES I J)) J)
       (REMAINDER X J))

Theorem.  PRIME.KEY.REWRITE (rewrite):
(IMPLIES (PRIME P)
         (EQUAL (EQUAL (REMAINDER (TIMES A B) P) 0)
                (OR (EQUAL (REMAINDER A P) 0)
                    (EQUAL (REMAINDER B P) 0))))
```

The latter is just a different statement of the lemma called PRIME.KEY in [2]. Another difference between the previously published library and the new one is that the lemma named EXP.TIMES in [2] is now named EXP.EXP and is used as a rewrite rule in the direction opposite that in [2]

```
Theorem.  EXP.EXP (rewrite):
(EQUAL (EXP (EXP I J) K) (EXP I (TIMES J K))).
```

The new library also contains a metafunction for canceling common addends on opposites sides of an equation, as described in [3]. Several lemmas in the old library, primarily about LESSP and PLUS, have been deleted because they are subsumed by the linear arithmetic procedure since incorporated into the system. In addition, we disabled the recursive definition of TIMES.

Readers interested in seeing the complete list of theorems in the initial library should write the authors.

6.2. Correctness of CRYPT

This section contains the formalization of the proof in Box 1.

1. Definition.

```
(CRYPT M E N)
   =
(IF
 (ZEROP E)
 1
 (IF
  (EVEN E)
  (REMAINDER (SQUARE (CRYPT M (QUOTIENT E 2) N))
             N)
  (REMAINDER
   (TIMES M
          (REMAINDER (SQUARE (CRYPT M (QUOTIENT E 2) N))
                     N))
   N)))
```

2. **Theorem.** TIMES.MOD.1 (rewrite):

```
(EQUAL (REMAINDER (TIMES X (REMAINDER Y N)) N)
       (REMAINDER (TIMES X Y) N))
```

3. **Theorem.** TIMES.MOD.2 (rewrite):

```
(EQUAL (REMAINDER (TIMES A (TIMES B (REMAINDER Y N)))
                  N)
       (REMAINDER (TIMES A B Y) N))
Hint:  Consider:
       TIMES.MOD.1 with {X/(TIMES A B)}
```

4. **Theorem.** CRYPT.CORRECT (rewrite):

```
(IMPLIES (NOT (EQUAL N 1))
         (EQUAL (CRYPT M E N)
                (REMAINDER (EXP M E) N)))
```

6.3. Miscellaneous Theorems

We now lay some ground work used throughout the rest of the proofs. These lemmas represent the "elementary properties of remainder and exponentiation" mentioned in the informal proofs. The first important result is event 7, which states that (a mod n)^{i} mod n is (a^{i}) mod n.

```
5.   Theorem.  TIMES.MOD.3 (rewrite):
     (EQUAL (REMAINDER (TIMES (REMAINDER A N) B) N)
            (REMAINDER (TIMES A B) N))

6.   Theorem.  REMAINDER.EXP/LEMMA (rewrite):
     (IMPLIES (EQUAL (REMAINDER Y A)
                     (REMAINDER Z A))
              (EQUAL (EQUAL (REMAINDER (TIMES X Y) A)
                            (REMAINDER (TIMES X Z) A))
                     T))

7.   Theorem.  REMAINDER.EXP (rewrite):
     (EQUAL (REMAINDER (EXP (REMAINDER A N) I) N)
            (REMAINDER (EXP A I) N))
```

We now proceed to teach the system several commonly used tricks. The first, event 8, shows the system that m^{i*j} mod n is 1 if m^{j} mod n is 1. To prove this obvious fact one must use the rewrite rules EXP.EXP and REMAINDER.EXP "against the grain" of the directed equality.

```
8.   Theorem.  EXP.MOD.IS.1 (rewrite):
     (IMPLIES (EQUAL (REMAINDER (EXP M J) P) 1)
              (EQUAL (REMAINDER (EXP M (TIMES I J)) P)
                     1))
     Hints:  Consider:
             EXP.EXP with {I/M, J/J, K/I}
             REMAINDER.EXP with {A/(EXP M J), N/P}
```

We next teach the system the trick of establishing (a mod p)=(b mod p) by considering whether p divides ¦a-b¦, and vice versa. We define PDIFFERENCE, the absolute value of the integer difference of two naturals, in terms of our function DIFFERENCE, which returns 0 when the subtrahend is larger than the minuend. We then prove the necessary theorems about PDIFFERENCE and, then at event 13, we tell the system henceforth not to expand the definition of PDIFFERENCE.

```
9.   Definition.
     (PDIFFERENCE A B)
        =
     (IF (LESSP A B)
         (DIFFERENCE B A)
         (DIFFERENCE A B))

10.  Theorem.  TIMES.DISTRIBUTES.OVER.PDIFFERENCE (rewrite):
     (EQUAL (TIMES M (PDIFFERENCE A B))
            (PDIFFERENCE (TIMES M A) (TIMES M B)))

11.  Theorem.  EQUAL.MODS.TRICK.1 (rewrite):
     (IMPLIES (EQUAL (REMAINDER (PDIFFERENCE A B) P)
                     0)
              (EQUAL (EQUAL (REMAINDER A P)
                            (REMAINDER B P))
                     T))

12.  Theorem.  EQUAL.MODS.TRICK.2 (rewrite):
     (IMPLIES (EQUAL (REMAINDER A P)
                     (REMAINDER B P))
              (EQUAL (REMAINDER (PDIFFERENCE A B) P)
                     0))
     Hint:  Disable DIFFERENCE.ELIM

13.  Disable PDIFFERENCE
```

We conclude this subsection by showing the system one last trick: to prove that (a mod p)=(b mod p), when p is prime, find an m such that p does not divide m and (m*a mod p)=(m*b mod p).

```
14.  Theorem.  PRIME.KEY.TRICK (rewrite):
     (IMPLIES (AND (EQUAL (REMAINDER (TIMES M A) P)
                          (REMAINDER (TIMES M B) P))
                   (NOT (EQUAL (REMAINDER M P) 0))
                   (PRIME P))
              (EQUAL (EQUAL (REMAINDER A P)
                            (REMAINDER B P))
                     T))
     Hints:  Consider:
             PRIME.KEY.REWRITE with {A/M, B/(PDIFFERENCE A B)}
             EQUAL.MODS.TRICK.2 with {A/(TIMES M A),
                                      B/(TIMES M B)}
```

6.4. Theorems 53 and 55

We now prove versions of Theorems 53 and 55 from [5] and observe trivial corollaries of each. Event 15 is used in the proof of event 16, which is used in the proof of Theorem 53.

```
15.  Theorem.  PRODUCT.DIVIDES/LEMMA (rewrite):
     (IMPLIES (EQUAL (REMAINDER X Z) 0)
              (EQUAL (REMAINDER (TIMES Y X) (TIMES Y Z))
                     0))
```

```
16.  Theorem.  PRODUCT.DIVIDES (rewrite):
     (IMPLIES (AND (EQUAL (REMAINDER X P) 0)
                   (EQUAL (REMAINDER X Q) 0)
                   (PRIME P)
                   (PRIME Q)
                   (NOT (EQUAL P Q)))
              (EQUAL (REMAINDER X (TIMES P Q)) 0))
```

```
17.  Theorem.  THM.53.SPECIALIZED.TO.PRIMES:
     (IMPLIES (AND (PRIME P)
                   (PRIME Q)
                   (NOT (EQUAL P Q))
                   (EQUAL (REMAINDER A P)
                          (REMAINDER B P))
                   (EQUAL (REMAINDER A Q)
                          (REMAINDER B Q)))
              (EQUAL (REMAINDER A (TIMES P Q))
                     (REMAINDER B (TIMES P Q))))
```

```
18.  Theorem.  COROLLARY.53 (rewrite):
     (IMPLIES (AND (PRIME P)
                   (PRIME Q)
                   (NOT (EQUAL P Q))
                   (EQUAL (REMAINDER A P)
                          (REMAINDER B P))
                   (EQUAL (REMAINDER A Q)
                          (REMAINDER B Q))
                   (NUMBERP B)
                   (LESSP B (TIMES P Q)))
              (EQUAL (EQUAL (REMAINDER A (TIMES P Q)) B)
                     T))
     Hint:  Consider:
            THM.53.SPECIALIZED.TO.PRIMES
```

```
19.  Theorem.  THM.55.SPECIALIZED.TO.PRIMES (rewrite):
     (IMPLIES (AND (PRIME P)
                   (NOT (EQUAL (REMAINDER M P) 0)))
              (EQUAL (EQUAL (REMAINDER (TIMES M X) P)
                            (REMAINDER (TIMES M Y) P))
                     (EQUAL (REMAINDER X P)
                            (REMAINDER Y P))))
```

20. Theorem. COROLLARY.55 (rewrite):

```
(IMPLIES (PRIME P)
         (EQUAL (EQUAL (REMAINDER (TIMES M X) P)
                       (REMAINDER M P))
                (OR (EQUAL (REMAINDER M P) 0)
                    (EQUAL (REMAINDER X P) 1))))
Hint:  Consider:
       THM.55.SPECIALIZED.TO.PRIMES with {Y/1}
```

6.5. The Pigeon Hole Principle

We are now on our way to Fermat's theorem and must state formally and prove the Pigeon Hole Principle. The amount of type-in for this theorem is relatively large. The reason is that the theorem is about concepts not already in the system's data base and about which the system knows nothing. Thus, we here have to build up a fair number of facts.

21. Definition.

```
(ALL.DISTINCT L)
   =
(IF (NLISTP L)
    T
    (AND (NOT (MEMBER (CAR L) (CDR L)))
         (ALL.DISTINCT (CDR L))))
```

22. Definition.

```
(ALL.LESSEQP L N)
   =
(IF (NLISTP L)
    T
    (AND (LEQ (CAR L) N)
         (ALL.LESSEQP (CDR L) N)))
```

23. Definition.

```
(ALL.NON.ZEROP L)
   =
(IF (NLISTP L)
    T
    (AND (NOT (ZEROP (CAR L)))
         (ALL.NON.ZEROP (CDR L))))
```

24. Definition.

```
(POSITIVES N)
   =
(IF (ZEROP N)
    NIL
    (CONS N (POSITIVES (SUB1 N))))
```

```
25.  Theorem.  LISTP.POSITIVES (rewrite):
     (EQUAL (LISTP (POSITIVES N))
            (NOT (ZEROP N)))

26.  Theorem.  CAR.POSITIVES (rewrite):
     (EQUAL (CAR (POSITIVES N))
            (IF (ZEROP N) 0 N))

27.  Theorem.  MEMBER.POSITIVES (rewrite):
     (EQUAL (MEMBER X (POSITIVES N))
            (IF (ZEROP X) F (LESSP X (ADD1 N))))

28.  Theorem.  ALL.NON.ZEROP.DELETE (rewrite):
     (IMPLIES (ALL.NON.ZEROP L)
              (ALL.NON.ZEROP (DELETE X L)))

29.  Theorem.  ALL.DISTINCT.DELETE (rewrite):
     (IMPLIES (ALL.DISTINCT L)
              (ALL.DISTINCT (DELETE X L)))

30.  Theorem.  PIGEON.HOLE.PRINCIPLE/LEMMA.1 (rewrite):
     (IMPLIES (AND (ALL.DISTINCT L)
                   (ALL.LESSEQP L (ADD1 N)))
              (ALL.LESSEQP (DELETE (ADD1 N) L) N))

31.  Theorem.  PIGEON.HOLE.PRINCIPLE/LEMMA.2 (rewrite):
     (IMPLIES (AND (NOT (MEMBER (ADD1 N) X))
                   (ALL.LESSEQP X (ADD1 N)))
              (ALL.LESSEQP X N))

32.  Theorem.  PERM.MEMBER (rewrite):
     (IMPLIES (AND (PERM A B) (MEMBER X A))
              (MEMBER X B))
```

The proof of the Pigeon Hole Principle we give employs an induction argument the system does not automatically construct. To tell it the induction argument we want used, we define a recursive function that mirrors the induction. The proof of the well-foundedness of the recursion justifies the induction scheme suggested by the function. Our first mechanical proof of the Pigeon Hole Principle used a machine generated induction, but required more preliminary work in the form of lemmas about ALL.LESSEQP, DELETE, and ALL.DISTINCT.

```
33. Definition.
    (PIGEON.HOLE.INDUCTION L)
        =
    (IF (LISTP L)
        (IF (MEMBER (LENGTH L) L)
            (PIGEON.HOLE.INDUCTION (DELETE (LENGTH L) L))
            (PIGEON.HOLE.INDUCTION (CDR L)))
        T)
```

```
34. Theorem.  PIGEON.HOLE.PRINCIPLE:
    (IMPLIES (AND (ALL.NON.ZEROP L)
                  (ALL.DISTINCT L)
                  (ALL.LESSEQP L (LENGTH L)))
             (PERM (POSITIVES (LENGTH L)) L))
    Hint:  Induct as for (PIGEON.HOLE.INDUCTION L).
```

We conclude this subsection by anticipating our use of the Pigeon Hole Principle by proving some elegant relations between permutations, products, the positives and the factorial function.

```
35. Theorem.  PERM.TIMES.LIST:
    (IMPLIES (PERM L1 L2)
             (EQUAL (TIMES.LIST L1)
                    (TIMES.LIST L2)))
```

```
36. Theorem.  TIMES.LIST.POSITIVES (rewrite):
    (EQUAL (TIMES.LIST (POSITIVES N))
           (FACT N))
```

```
37. Theorem.  TIMES.LIST.EQUAL.FACT (rewrite):
    (IMPLIES (PERM (POSITIVES N) L)
             (EQUAL (TIMES.LIST L) (FACT N)))
    Hint:  Consider:
           PERM.TIMES.LIST with {L1/(POSITIVES N), L2/L}
```

```
38. Theorem.  PRIME.FACT (rewrite):
    (IMPLIES (AND (PRIME P) (LESSP N P))
             (NOT (EQUAL (REMAINDER (FACT N) P) 0)))
    Hint:  Induct as for (FACT N).
```

6.6. Fermat's Theorem

This subsection is the formalization of the proof in Box 2.

39. Definition.

```
(S N M P)
   =
(IF (ZEROP N)
    NIL
    (CONS (REMAINDER (TIMES M N) P)
          (S (SUB1 N) M P)))
```

40. Theorem. REMAINDER.TIMES.LIST.S:

```
(EQUAL (REMAINDER (TIMES.LIST (S N M P)) P)
       (REMAINDER (TIMES (FACT N) (EXP M N))
                  P))
```

41. Theorem. ALL.DISTINCT.S/LEMMA (rewrite):

```
(IMPLIES (AND (PRIME P)
              (NOT (EQUAL (REMAINDER M P) 0))
              (NUMBERP N1)
              (LESSP N2 N1)
              (LESSP N1 P))
         (NOT (MEMBER (REMAINDER (TIMES M N1) P)
                      (S N2 M P))))
```

Hint: Induct as for (S N2 M P).

42. Theorem. ALL.DISTINCT.S (rewrite):

```
(IMPLIES (AND (PRIME P)
              (NOT (EQUAL (REMAINDER M P) 0))
              (LESSP N P))
         (ALL.DISTINCT (S N M P)))
```

43. Theorem. ALL.NON.ZEROP.S (rewrite):

```
(IMPLIES (AND (PRIME P)
              (NOT (EQUAL (REMAINDER M P) 0))
              (LESSP N P))
         (ALL.NON.ZEROP (S N M P)))
```

44. Theorem. ALL.LESSEQP.S (rewrite):

```
(IMPLIES (NOT (ZEROP P))
         (ALL.LESSEQP (S N M P) (SUB1 P)))
```

45. Theorem. LENGTH.S (rewrite):

```
(EQUAL (LENGTH (S N M P)) (FIX N))
```

46. Theorem. FERMAT.THM (rewrite):

```
(IMPLIES (AND (PRIME P)
              (NOT (EQUAL (REMAINDER M P) 0)))
         (EQUAL (REMAINDER (EXP M (SUB1 P)) P)
                1))
```

Hints: Consider:

```
PIGEON.HOLE.PRINCIPLE with {L/(S (SUB1 P) M P)}
REMAINDER.TIMES.LIST.S with {N/(SUB1 P)}
```

6.7. Invertibility of CRYPT

This subsection is the formalization of the proof in Box 3.

47. Theorem. CRYPT.INVERTS/STEP.1:

```
(IMPLIES
 (PRIME P)
 (EQUAL (REMAINDER (TIMES M (EXP M (TIMES K (SUB1 P))))
                   P)
        (REMAINDER M P)))
```

48. Theorem. CRYPT.INVERTS/STEP.1A (rewrite):

```
(IMPLIES
 (PRIME P)
 (EQUAL
  (REMAINDER
            (TIMES M
                   (EXP M (TIMES K (SUB1 P) (SUB1 Q))))
            P)
  (REMAINDER M P)))
Hint:  Consider:
       CRYPT.INVERTS/STEP.1 with {K/(TIMES K (SUB1 Q))}
```

49. Theorem. CRYPT.INVERTS/STEP.1B (rewrite):

```
(IMPLIES
 (PRIME Q)
 (EQUAL
  (REMAINDER
            (TIMES M
                   (EXP M (TIMES K (SUB1 P) (SUB1 Q))))
            Q)
  (REMAINDER M Q)))
Hint:  Consider:
       CRYPT.INVERTS/STEP.1
        with {P/Q, K/(TIMES K (SUB1 P))}
```

50. Theorem. CRYPT.INVERTS/STEP.2 (rewrite):

```
(IMPLIES
       (AND (PRIME P)
            (PRIME Q)
            (NOT (EQUAL P Q))
            (NUMBERP M)
            (LESSP M (TIMES P Q))
            (EQUAL (REMAINDER ED (TIMES (SUB1 P) (SUB1 Q)))
                   1))
       (EQUAL (REMAINDER (EXP M ED) (TIMES P Q))
              M))
```

51. **Theorem.** CRYPT.INVERTS:

```
(IMPLIES
      (AND (PRIME P)
           (PRIME Q)
           (NOT (EQUAL P Q))
           (EQUAL N (TIMES P Q))
           (NUMBERP M)
           (LESSP M N)
           (EQUAL (REMAINDER (TIMES E D)
                             (TIMES (SUB1 P) (SUB1 Q)))
                  1))
      (EQUAL (CRYPT (CRYPT M E N) D N) M))
```

7. References

1. N. Bourbaki. Elements of Mathematics. Addison Wesley, Readin, Massachusetts, 1968.

2. R. S. Boyer and J S. Moore. A Computational Logic. Academic Press, New York, 1979.

3. R. S. Boyer and J S. Moore. Metafunctions: Proving Them Correct and Using Them Efficiently as New Proof Procedures. In The Correctness Problem in Computer Science, R. S. Boyer and J S. Moore, Eds., Academic Press, London, 1981.

4. K. Godel. On Formally Undecidable Propositions of Principia Mathematica and Related Systems. In From Frege to Goedel, J. van Heijenoort, Ed.,Harvard University Press, Cambridge, Massachusetts, 1967.

5. G. H. Hardy and E. M. Wright. An Introduction to the Theory of Numbers. Oxford University Press, 1979.

6. D. E. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In Computational Problems in Abstract Algebras, J. Leech, Ed.,Pergamon Press, Oxford, 1970, pp. 263-297.

7. D. E. Knuth. The Art of Computer Programming. Volume 1/ Fundamental Algorithms. Addison-Wesley Publishing Co., Reading, MA, 1973.

8. D. E. Knuth. The Art of Computer Programming. Volume 2/ Seminumerical Algorithms. Addison-Wesley Publishing Co., Reading, MA, 1981.

9. J S. Moore. "A Mechanical Proof of the Termination of Takeuchi's Function." Information Processing Letters 9, 4 (1979), 176-181.

10. R. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM 21, 2 (1978), 120-126.

8. Distribution List

Defense Documentation Center (12 copies)
Cameron Station
Alexandria, VA 22314

Naval Research Laboratory (6 copies)
Technical Information Division
Code 2627
Washington, D.C. 20375

Office of Naval Research (2 copies)
Information Systems Program (437)
Arlington, VA 22217

Office of Naval Research
Code 200
Arlington, VA 22217

Office of Naval Research
Code 455
Arlington, VA 22217

Office of Naval Research
Code 458
Arlington, VA 22217

Office of Naval Research
Eastern/Central Regional Office
Bldg 114, Section D
666 Summer Street
Boston, MA 02210

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605

Office of Naval Research
Western Regional Office
1030 East Green Street
Pasadena, CA 91106

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

Naval Ocean Systems Center
Advanced Software Technology Div.
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner
Naval Ship Research
& Development Center
Computation and Mathematics Dept.
Bethesda, MD 20084

Captain Grace M. Hopper (008)
Naval Data Automation Command
Washington Navy Yard
Building 166
Washington, D.C. 20374

END

FILMED

9-83

DTIC